Scheduling Constrained-Deadline Sporadic Parallel Tasks Considering Memory Contention

Björn Andersson, Dionisio de Niz, Hyoseung Kim, Mark Klein, Ragunathan (Raj) Rajkumar
Carnegie Mellon University

***Abstract*—Consider constrained-deadline sporadic tasks scheduled on a multiprocessor where (i) each task is characterized by its execution requirement, deadline, and minimum inter-arrival time, (ii) each task generates a sequence of jobs, (iii) the execution requirement of a job and its potential for parallel execution is described by one or many stages with a stage having one or many segments and different segments in the same stage can execute in parallel and a segment is only allowed to start execution if all segments of previous stages have finished execution, and (iv) there is contention for shared resources in the memory system (cache eviction, reordering in memory controller, memory bus contention). We present an algorithm that (i) performs schedulability testing for tasks scheduled with global-Earliest-Deadline-First (gEDF), (ii) configures the virtual-to-physical address translation so that a cache block fetched to the last-level cache by one task cannot be evicted by another task, (iii) configures the virtual-to-physical address translation to attempt to eliminate the extra execution time caused by the reordering effect in the memory controller and if this is not possible, then the reordering effect is considered in the schedulability analysis, and (iv) considers the effect of contention for the memory bus. Our solution is based on formulating this problem as a Mixed-Integer Linear Program (MILP). We have implemented a tool based on this theory and validated its output against measurements on a real computer.**

I. Introduction

Multicore processors are the norm today. The trend is that the number of processors on a chip increases exponentially while the clock frequency stays constant. And software practitioners are under pressure to deliver improved functionality which has increased the execution requirements. This trend makes it increasingly common in real-time systems that a job has execution requirement so large that executing it sequentially causes a deadline miss and hence, the only way for a job to meet its deadline is to perform some execution in parallel. Some software is inherently sequential, however, so a software system typically consists of parts that can execute in parallel and parts that cannot. This brings the challenge:

> **C1.** Schedule software where some parts can execute in parallel so that all deadlines are met and prove before run-time that their deadlines are met.

Timing of software executing on a COTS multicore processor depends not only on the processor scheduler but also on contention for shared resources in the memory system. This includes (i) the last-level cache shared between processors, (ii) the row buffer in each memory bank storing the most recently accessed row, and (iii) the memory bus (the bus between the memory controller and DRAM memory modules). A cache memory is typically organized as a set of cache sets where certain bits of the physical address of a memory access determine which cache set the memory access should use. Hence, if the virtual-to-physical address translation is set up so that for the physical addresses generated, it holds that no two memory accesses of different tasks use the same cache set, then it is guaranteed that a cache block fetched to the cache by one task cannot be evicted by another task. Also, DRAM memories are typically organized as a set of banks with each bank having multiple rows and each bank having one row buffer which stores the data of the most recently accessed row. When a memory access experiences a miss in the shared cache, (i) precharging is performed, that is, the data in the row buffer is written back to its row in the memory bank and then (ii) the memory access activates a row in a memory bank (the memory bank is indicated by certain bits in the physical address of the memory access and the row is indicated by other bits) so this row is loaded in the row buffer of the memory bank and then (iii) the memory access reads data from this buffer and transfers the data to the processor (if the memory access is a load) or writes data to this row buffer (if the memory access is a store). If the row needed for a memory access is already loaded in the row buffer, then precharge and activate are not performed and hence execution is faster. For this reason, memory controllers reorder memory accesses so that memory accesses to the row that is in the row buffer get ahead in certain queues in the memory controller. Consequently, a memory access can be delayed because other memory accesses, of other tasks, get ahead in the queue (reordering effect). Hence, if the virtual-to-physical address translation is set up so that for the physical addresses generated by tasks, it holds that no two of them access the same bank, then it is guaranteed that no task can suffer from this reordering effect. In addition, a memory access can also be delayed because other accesses use the memory bus. This brings the challenges:

> **C2.** Configure the virtual-to-physical address translation so that a cache block fetched to the last-level cache by one task cannot be evicted by another task.

> **C3.** Configure the virtual-to-physical address translation so that reordering of memory accesses from different tasks are avoided and if they do occur, then the schedulability analysis computes an upper bound on the extra execution time due to reordering.

> **C4.** Compute an upper bound on the extra execution time caused by processors sharing the memory bus.

The research literature offers solutions for each of these challenges (see Table I). Unfortunately, the research literature offers no solution for *all* these challenges.

Therefore, in this paper, we present a solution for *all* these challenges. We assume global-EDF (gEDF) is used and

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **01 OCT 2014**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Scheduling Constrained-Deadline Sporadic Parallel**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Bjorn A. Andersson Dionisio de Niz, Hyoseung Kim, Mark Klein, Ragunathan (Raj) Rajkumar**		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**		
13. SUPPLEMENTARY NOTES **The original document contains color images.**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**10**	

Solution	Addresses challenges			
	C1	C2	C3	C4
[16, 9, 26, 12, 7, 11, 6, 21, 3, 4, 1, 10, 15, 5, 17, 23, 2, 29]	Yes	No	No	No
[20, 30]	No	Yes	No	No
[18, 24]	No	No	Yes	No
[28]	No	Yes	Yes	No
[25, 27, 8, 19, 22, 32, 31, 13]	No	No	No	Yes
This paper	Yes	Yes	Yes	Yes

TABLE I: Summary of the state of art.

consider a previously known [1] schedulability test for it; we choose gEDF and this schedulability test because among the schedulers and schedulability tests available for tasks with potential parallelism, gEDF with our chosen schedulability test offers the best performance bound [1]. We reformulate this schedulability test as a Mixed-Integer Linear Program (MILP) and extend this formulation so that it (i) configures the virtual-to-physical address translation so that a cache block fetched to the last-level cache by one task cannot be evicted by another task, (ii) configures the virtual-to-physical address translation to attempt to eliminate the extra execution time caused by the reordering effect in the memory controller, and if this is not possible, the reordering effect is considered in the schedulability analysis, and (iii) considers the effect of contention for the memory bus.

The remainder of this paper is organized as follows. Section II presents the system model we use. Section III adapts a previously known schedulability test for gEDF to a MILP formulation. Section IV presents additional constraints that express an upper bound on the execution time of a segment due to memory contention and how it depends on memory mapping, and also express other constraints. Section V puts it all together as a solution for all the four challenges. Then follow discussions and conclusions.

II. System Model

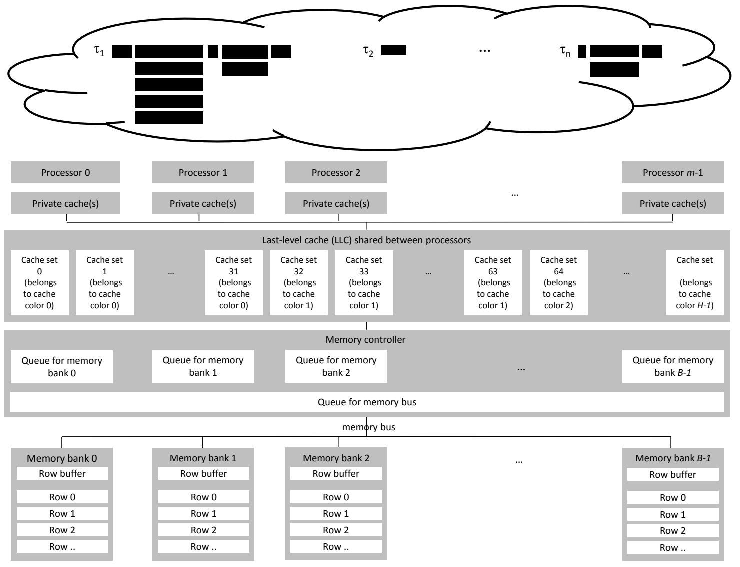
Fig. 1 illustrates the model we consider. We consider a system with (i) a computer with m processors of speed s and (ii) a software system described as a taskset τ. A task τ_i in τ is characterized by T_i, D_i, $\mathrm{nstages_i}$, $\mathrm{nseg_{i,j}}$, and $C_{i,j}$ with the interpretation that τ_i generates a sequence of jobs where the arrival times of two consecutive jobs of τ_i are separated by at least T_i and a job of τ_i needs to finish execution by the absolute deadline of the job (the absolute deadline of a job of τ_i is D_i time units after its arrival) and execution requirement is described with stages where $\mathrm{nstages_i}$ denotes the number of stages of a job of τ_i and $\mathrm{nseg_{i,j}}$ denotes the number of segments of the j^{th} stage of a job of τ_i. Let $C_{i,j}$ denote an upper bound on the execution requirement of a segment of the j^{th} stage of τ_i (explained later in this section). A job executing contiguously for Δ time units performs $\Delta \times s$ units of execution. We assume $\forall \tau_i \in \tau : D_i \leq T_i$ — such tasksets are called *constrained-deadline sporadic tasksets*.

When a job of task τ_i arrives, all the $\mathrm{nseg_{i,1}}$ segments of the 1^{st} stage of task τ_i become eligible for execution. For each $j \geq 2$, at the time when all the $\mathrm{nseg_{i,j-1}}$ segments of the $(j-1)^{\mathrm{th}}$ stage of task τ_i have finished, all the $\mathrm{nseg_{i,j}}$ segments of the j^{th} stage of task τ_i become eligible for execution. A segment becomes non-eligible when it has finished execution. A job of task τ_i finishes when all the $\mathrm{nseg_{i,nstages_i}}$ segments of the $\mathrm{nstages_i}^{\mathrm{th}}$ stage of this job have finished.

gEDF assigns high priority to jobs with early absolute deadlines and a segment inherits the priority of the job it belongs to. At each instant, if at most m segments are eligible for execution at this instant, then all of them execute at this instant; if $m+1$ or more segments are eligible for execution, then the m highest priority segments at this instant are selected for execution at this instant. A taskset τ is *gEDF schedulable* on a computer with m processors of speed s if for each jobset that τ can generate, for each schedule that gEDF can generate for this jobset, it holds that all deadlines are met.

Each segment of a stage of a task has a virtual address space. The virtual address space is organized into pages of size PAGESIZE. (For example, for x86, PAGESIZE=4096 bytes.) The memory footprint of a segment of the j^{th} stage of τ_i is at most $\mathrm{np}_{i,j}$ pages. Each page is associated with a range of virtual addresses. A virtual address is mapped to a physical address as follows. The $\log_2 \mathrm{PAGESIZE}$ least significant bits of the virtual address are copied to the least significant bits of the physical address. (These least significant bits of the physical address are called *frame offset*.) The other bits of the virtual address are called *page index* and these are translated to a *frame index* and the bits of the frame index are copied to the most significant bits of the physical address. $\mathrm{sharedframes}$ denotes a set of 8-tuples so that for each 8-tuple $\langle i', j', g', p', i'', j'', g'', p'' \rangle$ it is required that page p' of the g'^{th} segment of the j'^{th} stage of $\tau_{i'}$ is mapped to the same frame as page p'' of the g''^{th} segment of the j''^{th} stage of $\tau_{i''}$. We assume that for such 8-tuples $\tau_{i'} = \tau_{i''}$ (because otherwise cache coloring, as we will see, does not work).

In our previous work [13], we presented and validated a model of the memory system of typical COTS multicore processor based systems. In this paper, we use a model that improves on that model by having a more fine-grained description of memory accesses. Our model is as follows. The last-level cache (LLC) is shared between processors. This cache is organized as a set of cache sets where certain bits in the physical address determine which cache set a memory access is associated with. Some of these bits are part of the frame index and some are part of the frame offset. When a memory access experiences a miss in LLC, the memory access is passed on to the memory controller and identifies which memory bank the memory access is associated with (certain bits in the frame index determine this) and which row in this memory bank it is associated with (other bits in the frame index determine this) and it is inserted in a queue for memory accesses to this memory bank. The queuing discipline First-Ready-First-In-First-Out (FR-FIFO) is used. With this queuing discipline, FIFO is used but with the following exceptions (i) a memory access can be prevented from being performed at certain instants because there are DRAM timing parameters which state that a certain part of a DRAM access must wait until a certain timing requirement (based on previous memory accesses) is satisfied and (ii) elements in the queue of a memory bank can be reordered so that a memory access gets to the head of the queue when this memory access is associated with the row that is currently loaded in the row buffer. When a memory access gets to the head of the queue of the memory bank, it contends for the memory bus with memory accesses



τ_1
τ_2
...
τ_n
Processor 0
Processor 1
Processor 2
Processor *m*-1
Private cache(s)
Private cache(s)
Private cache(s)
...
Private cache(s)
Last-level cache (LLC) shared between processors
Cache set 0 (belongs to cache color 0)
Cache set 1 (belongs to cache color 0)
...
Cache set 31 (belongs to cache color 0)
Cache set 32 (belongs to cache color 1)
Cache set 33 (belongs to cache color 1)
...
Cache set 63 (belongs to cache color 1)
Cache set 64 (belongs to cache color 2)
...
Cache set (belongs to cache color *H-1*)
Memory controller
Queue for memory bank 0
Queue for memory bank 1
Queue for memory bank 2
...
Queue for memory bank *B-1*
Queue for memory bus
memory bus
Memory bank 0
Memory bank 1
Memory bank 2
...
Memory bank *B-1*
Row buffer
Row 0
Row 1
Row 2
Row ..

Fig. 1: The model we consider.

of other memory banks. When a memory access is granted the memory bus, the memory access *precharges* its associated memory bank (that is, the data in the row buffer is written back to its row in the memory bank) and then the memory access *activates* its associated row in its associated memory bank (that is, the data in this row is loaded to the row buffer) and finally it transfers data (from the row buffer of the memory bank to the memory controller if the memory access is a load; the other direction if it is a store). If the row associated with the memory access is already in the row buffer then precharge and activate are not performed.

$C_{i,j}(\text{map})$ denotes an upper bound on the execution requirement of a segment in the j^{th} stage of τ_i for the case that this segment does not experience contention for resources in the memory system from other segments and map is the memory mapping of all tasks in the system. $\text{MA}_{i,j,p}(\text{map})$ denotes an upper bound on the number of memory accesses reaching the memory controller of page p of a segment in the j^{th} stage of τ_i for the case that this segment does not experience contention for resources in the memory system from other segments and map is the memory mapping of all tasks in the system.

In today's processors, typically, the bits in the physical address from which the cache set index of LLC is obtained overlaps with the bits that determine the frame index (see [28]). Also, in today's processors, typically, the bits in the the physical address from which the bank index is obtained overlaps with the bits that determine the frame index (see [28]). Therefore, one can partition memory frames of physical memory into cache colors so if two memory accesses belong to different frames and these two memory frames belong to two different cache colors, then it holds that one memory access cannot evict a cache block fetched to LLC by the another memory access. One can also partition memory frames of physical memory into bank colors so if two memory accesses belong to different frames and these two memory frames belong to two different bank colors, then it holds that one memory access cannot evict a row in a memory bank that another memory access has loaded. H denotes the number of cache colors and B denotes the number of bank colors. MEMCAP denotes the amount of physical memory in the computer, measured in the number of frames. Some recent multicore chips use sliced LLC; such chips prevent us from using 100% of the main memory when using cache partitioning [14]. For this reason, let HWSHARE denote the share of physical memory that we may use. Let $\text{CAP} = \text{HWSHARE} \times \text{MEMCAP}/(H \times B)$. (In a typical x86 computer today, $\text{MEMCAP} = 2^{31}/4096 = 2^{19}$,$H = 32$,$B = 16$, $\text{HWSHARE} = 1/4$ and this yields $\text{CAP} = (1/4) \times 2^{19}/(32 \times 16) = 2^8$)

Let $C_{i,j}$ be a value such that $\forall \text{map} : C_{i,j}(\text{map}) \leq C_{i,j}$. Let $\text{MA}_{i,j,p}$ be a value such that $\forall \text{map} : \text{MA}_{i,j,p}(\text{map}) \leq \text{MA}_{i,j,p}$. In practice, if the memory mapping map is known, then it is possible to obtain $C_{i,j}(\text{map})$ and $\text{MA}_{i,j,p}(\text{map})$ (e.g. using a worst-case execution-time analysis tool) but obtaining $C_{i,j}$ and $\text{MA}_{i,j,p}$ is very expensive because they describe behavior of the software for *all possible memory mappings* of the system. Even if $C_{i,j}$ and $\text{MA}_{i,j,p}$ are obtained, it can happen that our algorithm selects an abstract memory mapping o and we choose a memory mapping map that is compatible with o and that $C_{i,j}$ is much higher than $C_{i,j}(\text{map})$ (and analogously for $\text{MA}_{i,j,p}(\text{map})$). This would result in large pessimism. We will discuss (in Section VI) how to deal with these issues. For now, assume $C_{i,j}$ and $\text{MA}_{i,j,p}$ are known.

We assume (as do many previous studies [31, 22, 19]) that a processor is stalled when it waits for memory. We use some notation from [13], namely, the following:

$$L^{\text{PRE}}_{\text{inter}} = t_{\text{CK}} \tag{8}$$

$$L^{ACT}_{\text{inter}} = \max(t_{\text{RRD}}, t_{\text{FAW}} - 3 \times t_{\text{RRD}}) \times t_{\text{CK}} \tag{9}$$

$$L^{\text{RW}}_{\text{inter}} = \max(\text{WL} + \text{BL}/2 + t_{\text{WTR}}, \text{CL} + \text{BL}/2 + 2 - \text{WL}) \times t_{CK} \tag{10}$$

$$L_{\text{inter}} = L^{\text{PRE}}_{\text{inter}} + L^{ACT}_{\text{inter}} + L^{\text{RW}}_{\text{inter}} \tag{11}$$

$$\begin{aligned} L_{\text{conf}} &= t_{\text{RP}} + t_{\text{RCD}} + \\ &\quad \max(\text{CL} + \text{BL}/2 + 2, \text{WL} + \text{BL}/2 + \max(t_{\text{WTR}}, t_{\text{WR}})) \times t_{\text{CK}} \end{aligned} \tag{12}$$

$$\begin{aligned} L_{\text{conhit}}(x) &= (\lceil x/2 \rceil \times (\text{WL} + \text{BL}/2 + t_{\text{WTR}}) + \\ &\quad \lfloor x/2 \rfloor \times \text{CL} + \max(t_{\text{WR}} - t_{\text{WTR}})) \times t_{\text{CK}} \end{aligned} \tag{13}$$

$$C_i \stackrel{\text{def}}{=} \sum_{j=1}^{\text{nstages}_i} (\text{nseg}_{\text{i,j}} \times C_{i,j}) \qquad \eta_i \stackrel{\text{def}}{=} \sum_{j=1}^{\text{nstages}_i} \left(\lceil \frac{\text{nseg}_{\text{i,j}}}{m} \rceil \times C_{i,j} \right) \tag{1}$$

$$\text{WJ}(\tau_i, t, s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if t} < 0 \\ \text{WJS}(i, t, 1, s) & \text{if } 0 \leq \text{t} < \frac{\eta_i}{s} \\ C_i & \text{if } \frac{\eta_i}{s} \leq \text{t} \end{cases} \qquad \text{bsp}_{i,j} \stackrel{\text{def}}{=} \frac{C_{i,j}}{s} \times \lfloor \frac{\text{nseg}_{i,j}}{m} \rfloor \qquad \text{sp}_{i,j} \stackrel{\text{def}}{=} \frac{C_{i,j}}{s} \times \lceil \frac{\text{nseg}_{i,j}}{m} \rceil \tag{2}$$

$$\text{WJS}(i, t, j, s) \stackrel{\text{def}}{=} \begin{cases} t \times m \times s & \text{if } 0 \leq t < \text{bsp}_{i,j} \\ \text{bsp}_{i,j} \times m \times s + (t - \text{bsp}_{i,j}) \times (\text{nseg}_{i,j} \bmod m) \times s & \text{if } \text{bsp}_{i,j} \leq t < \text{sp}_{i,j} \\ C_{i,j} \times \text{nseg}_{i,j} + \text{WJS}(i, t - \text{sp}_{i,j}, j+1, s) & \text{if } \text{sp}_{i,j} \leq \text{t} \end{cases} \tag{3}$$

$$\text{ffdbf}(\tau_i, t, v, s) \stackrel{\text{def}}{=} \lfloor \frac{t}{T_i} \rfloor \times C_i + C_i - \text{WJ}(\tau_i, (D_i - (t \bmod T_i)) \times v, s) \tag{4}$$

$$\text{h}(\tau, m, s, \sigma, t) \stackrel{\text{def}}{=} (\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t, \frac{\sigma}{s}, s) \leq ((m - (m-1) \times \frac{\sigma}{s}) \times t \times s)) \tag{5}$$

$$\text{f}(\tau, m, s) \stackrel{\text{def}}{=} \big(\exists \sigma \text{ such that } (\sigma \geq \max_{\tau_i \in \tau} \frac{\eta_i}{D_i}) \wedge (\forall t \text{ such that } t \geq 0 : \text{h}(\tau, m, s, \sigma, t))\big) \tag{6}$$

$$\text{f}(\tau, m, s) \Rightarrow \tau \text{ is gEDF schedulable on a computer with } m \text{ processors of speed } s \text{ for the case that tasks do not experience memory contention} \tag{7}$$

Fig. 2: Previously known schedulability analysis for gEDF scheduling of parallel tasks.

The first is the time required for precharge; the second is the time required for activate; the third is the time for data transfer. L_{conf} denotes row-conflict service time and $L_{\text{conhit}}(x)$ is a function which describes the time it takes to serve x consecutive memory accesses to the same row in the same memory bank if the row was already activated. These parameters can be obtained from DRAM datasheets — see [13]. P denotes the least common multiple of T_i values. $\text{DMAX} = \max_{\tau_i \in \tau} D_i$. N_{re} is a parameter which describes the limit that the hardware imposes on reordering (to be discussed later). We define $\text{UBNOMR} = \sum_{\tau_i'} (\max_{j' \in [1, \text{nstages}_{i'}]} \text{seg}_{i',j'})$ meaning upper bound on the number of outstanding memory requests. We also define: $\text{LIM1} = \min(m-1, \text{UBNOMR}-1)$ and $\text{LIM2} = \min(m-1, \text{UBNOMR}-1) + N_{\text{re}}$.

Tasks typically perform execution and access memory in an initialization phase which does not have real-time requirements. This execution and memory accesses are not considered as a job but the pages accessed needs to be mapped to memory frames. Therefore, INO indicates the the number of pages accessed during initialization.

III. Schedulability analysis for gEDF of parallel tasks

Previous work [1] provided a schedulability test for this problem for the special case that contention for resources in the memory system does not occur. Fig. 2 shows this schedulability test. We will now discuss how to modify this schedulability test slightly and then rewrite as MILP.

Let us choose a value of K that is a positive integer (e.g. $K = 20$). In the schedulability test expressed by Fig. 2, check only those σ such that there is a $k \in \{1, 2, \ldots, K\}$ such that $\sigma = (k/K) \times s$. This yields the following schedulability test with a slight increase in pessimism:

$$\text{h}^*(\tau, m, s, k, K, t) \stackrel{\text{def}}{=} (\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t, \frac{k}{K}, s) \leq (m - (m-1) \times \frac{k}{K}) \times t \times s)$$

$$\text{f}^*(\tau, m, s, K) \stackrel{\text{def}}{=} \big(\exists k \in \{1, 2, \ldots, K\} \text{ such that } (\frac{k \times s}{K} \geq \max_{\tau_i \in \tau} \frac{\eta_i}{D_i}) \wedge (\forall t \text{ such that } t \geq 0 : \text{h}^*(\tau, m, s, k, K, t))\big)$$

$$\text{f}^*(\tau, m, s, K) \Rightarrow \tau \text{ is gEDF schedulable on a computer with } m \text{ processors of speed } s \text{ for the case that tasks do not experience memory contention}$$

Clearly, $t = \lfloor t/P \rfloor \times P + t \bmod P$. Thus $\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t, \frac{k}{K}, s) = \lfloor t/P \rfloor \times P \times (\sum_{\tau_i \in \tau} C_i/T_i) + \sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t \bmod P, \frac{k}{K}, s)$. Consequently when evaluating $(\forall t \text{ such that } t \geq 0 : \text{h}^*(\tau, m, s, k, K, t))$, it is only necessary to consider values of t that are at most P. Hence, $\text{f}^*(\tau, m, s, K)$ is true if and only if there is an assignment of values satisfying: $\sum_{k=1}^{K} \text{wi}_k \geq 1$ and

$$\forall k \in [1, K] : \text{wi}_k \in \{0, 1\}$$

$$\forall \langle i, k \rangle \text{ such that } (\tau_i \in \tau) \wedge (k \in [1, K]) : (\text{wi}_k = 1) \Rightarrow (\eta_i \leq \frac{s \times k \times D_i}{K})$$

$$\forall k \text{ such that } k \in [1, K] : (\text{wi}_k = 1) \Rightarrow$$
$$(\forall t \text{ such that } t \in [0, P] : \text{h}^*(\tau, m, s, k, K, t))$$

Observe that the left-hand side of the inequality defining $\text{h}^*(\tau, m, s, k, K, t)$ is a piecewise linear function of t and the right-hand side of the inequality defining $\text{h}^*(\tau, m, s, k, K, t)$ is a linear function of t. Hence, when evaluating $(\forall t \text{ such that } t \in [0, P] : \text{h}^*(\tau, m, s, k, K, t))$ it is only necessary to evaluate $\text{h}^*(\tau, m, s, k, K, t)$ for the following values of t: (i) values of t such that the derivative of the piecewise linear function changes, (ii) $t = P$, and (iii) $t = 0$. With respect to (iii), note that $\text{h}^*(\tau, m, s, k, K, 0)$ is true and hence it does not need to be checked. With respect to (ii), note that $\text{h}^*(\tau, m, s, k, K, P)$ can be rewritten as $((\sum_{\tau_i \in \tau} C_i/T_i) \leq (m-(m-1) \times (k/K)) \times s)$. With respect to (i), note that for t such that there is a positive integer q' and a task $\tau_{i'} \in \tau$ such that $t = (q'-1) \times T_{i'} + D_{i'} - (\eta_{i'}/s) \times (K/k)$, the above mentioned derivative changes but this t is dominated by other t:s in the condition and hence, this t does not need to be checked. Hence, $\mathrm{f}^*(\tau, m, s, K)$ is true if and only if there is an assignment of values satisfying: $\sum_{k=1}^{K} \mathrm{wi}_k \geq 1$ and

$$\forall k \in [1, K] : \mathrm{wi}_k \in \{0, 1\}$$
$$\forall \langle i, k \rangle \text{ such that } (\tau_i \in \tau) \wedge (k \in [1, K]) : (\mathrm{wi}_k = 1) \Rightarrow (\eta_i \leq \frac{s \times k \times D_i}{K})$$
$$\forall \langle i', q', j', f', k \rangle \text{ such that } (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge$$
$$(j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0, 1\}) \wedge (k \in [1, K]) : t_{i',q',j',f',k} =$$
$$(q' - 1) \times T_{i'} + D_{i'} - ((\sum_{j'' \in [1, j']} \mathrm{sp}_{i',j''}) + f' \times \mathrm{bsp}_{i',j'+1}) \times \frac{K}{k}$$
$$\forall \langle i', q', j', f', k \rangle \text{ such that } (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge$$
$$(j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0, 1\}) \wedge (k \in [1, K]) : (\mathrm{wi}_k = 1) \Rightarrow$$
$$(\sum_{\tau_i \in \tau} \mathrm{ffdbf}(\tau_i, t_{i',q',j',f',k}, \frac{k}{K}, s)$$
$$\leq (m - (m - 1) \times \frac{k}{K}) \times t_{i',q',j',f',k} \times s)$$
$$\forall k \text{ such that } k \in [1, K] : (\mathrm{wi}_k = 1) \Rightarrow$$
$$((\sum_{\tau_i \in \tau} C_i/T_i) \leq (m - (m - 1) \times (k/K)) \times s)$$

We will now rewrite $\mathrm{ffdbf}(\tau_i, t_{i',q',j',f',k}, \frac{k}{K}, s)$ to a form closer to MILP. Define $I_{i,q,i',q',j',f',k}$ so that $I_{i,q,i',q',j',f',k} = 1$ if $\lfloor t_{i',q',j',f',k}/T_i \rfloor = q$; otherwise $I_{i,q,i',q',j',f',k} = 0$. Define $r_{i,i',q',j',f',k} = t_{i',q',j',f',k} \bmod T_i$. Define $\mathrm{aj}_{i,i',q',j',f',k}$ so that $(I_{i,q,i',q',j',f',k} = 1) \Rightarrow (\mathrm{aj}_{i,i',q',j',f',k} = (q+1) \times C_i)$. Then, using (4), rewrite $\mathrm{ffdbf}(\tau_i, t_{i',q',j',f',k}, \frac{k}{K}, s)$ as:

$$\mathrm{aj}_{i,i',q',j',f',k} - \mathrm{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$$

We introduce wjf,wjsf,wjss, and wjt as:

1) $\mathrm{wjf}_{i,i',q',j',f',k} = 1$ means that when $\mathrm{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the first case in the definition of WJ is taken; otherwise $\mathrm{wjf}_{i,i',q',j',f',k} = 0$.
2) $\mathrm{wjsf}_{i,j,i',q',j',f',k} = 1$ means that when $\mathrm{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the second case in the definition of WJ is taken and recursion is performed in WJS in which stage j is the last entire stage covered and the first case in (3) is taken; otherwise $\mathrm{wjsf}_{i,j,i',q',j',f',k} = 0$.
3) $\mathrm{wjss}_{i,j,i',q',j',f',k} = 1$ means that when $\mathrm{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the second case in the definition of WJ is taken and recursion is performed in WJS in which stage j is the last entire stage covered and the second case in (3) is taken; otherwise $\mathrm{wjss}_{i,j,i',q',j',f',k} = 0$.
4) $\mathrm{wjt}_{i,i',q',j',f',k} = 1$ means that when $\mathrm{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the third case in the definition of WJ is taken; otherwise $\mathrm{wjt}_{i,i',q',j',f',k} = 0$.

Elaborating on this yields that $\mathrm{f}^*(\tau, m, s, K)$ is true if and only if there is an assignment of values to variables such that the constraints in Fig. 3 are satisfied. In Fig. 2, $C_{i,j}$ denotes the upper bound on the execution requirement of a segment in the j^{th} stage of task τ_i but in Fig. 3 $\mathrm{cu}_{i,j}$ denotes this. ($\mathrm{cu}_{i,j}$ means execution requirement that we will use.)

IV. Memory Contention

Previous work [13] provided a method for computing an upper bound on the response time of a task considering contention for resources in the memory system. That method assumes fixed-priority preemptive non-migrative scheduling and integrates the memory contention analysis in the schedulability analysis. In this section, we will adapt this memory contention analysis (i) to compute an upper bound on the extra execution time of a segment of a single job of a task and do it without assuming any specific processor scheduling algorithm and (ii) expressing it on a form easily translatable to MILP.

Let $\mathrm{cm}_{i,j,g}$ denote an upper bound on the execution requirement of the g^{th} segment of the j^{th} stage of task τ_i considering contention for resources in the memory system (the extra execution of this contention is considered to be part of the execution requirement). Also, recall that $\mathrm{cu}_{i,j}$ was defined in the previous section. We will now redefine it. Let $\mathrm{cu}_{i,j}$ denote an upper bound on the execution requirement of a segment of the j^{th} stage of task τ_i considering contention for resources in the memory system (the extra execution of this contention is considered to be part of the execution requirement). Hence:

$$\mathrm{cu}_{i,j} = \max_{g \in [1, \mathrm{nseg}_{i,j}]} \mathrm{cm}_{i,j,g} \tag{34}$$

(47),(48),(49) express (34) as MILP.

Let $\mathrm{o}_{i,j,g,p,h,b} = 1$ indicate that the page with page index p of the g^{th} segment of the j^{th} stage of task τ_i is mapped to a memory frame with cache color h and bank color b; otherwise $\mathrm{o}_{i,j,g,p,h,b} = 0$. Clearly, a page can only be mapped to one frame and one frame belongs to exactly one cache color and one bank color. Hence, each page belongs to exactly one combination of cache and bank color. This yields (41). Also, if a cache color and bank color is given then the number of pages that can be mapped to this combination of cache color and bank color cannot exceed its amount of physical memory. In order to express this, let $\mathrm{GS}_{i,j,g,p}$ be a constant that indicates how many pages maps to the same frame as page p of the g^{th} segment of the j^{th} stage of τ_i maps to. For normal pages, it holds that $\mathrm{GS}_{i,j,g,p} = 1$ but if a page maps to a shared frame then $\mathrm{GS}_{i,j,g,p}$ is larger. $\mathrm{GS}_{i,j,g,p}$ can be computed as follows. Form a graph, with one vertex for each each $\langle i, j, g, p \rangle$ and there is an edge between two vertices $\langle i', j', g', p' \rangle$ and $\langle i'', j'', g'', p'' \rangle$ if $\langle i', j', g', p', i'', j'', g'', p'' \rangle \in \mathrm{sharedframes}$. Compute the connected components of the graph. Then, for $\langle i, j, g, p \rangle$, let $\mathrm{GSVS}_{i,j,g,p}$ denote the set of vertices in the connected component to which the vertex corresponding to $\langle i, j, g, p \rangle$ belong. let $\mathrm{GSTS}_{i,j,g,p}$ denote the set of tuples that correspond to $\mathrm{GSVS}_{i,j,g,p}$. Let $\mathrm{GS}_{i,j,g,p}$ denote the cardinality of $\mathrm{GSTS}_{i,j,g,p}$.

With $\mathrm{GS}_{i,j,g,p}$, the limited memory capacity can be expressed by (42). In addition, the requirement on shared frames, expressed by the set $\mathrm{sharedframes}$, yields (50). $\mathrm{ino}_{h,b}$ indicates the the number of pages accessed during initialization that maps to frames of which belong to cache color h and bank color b.

For a pair of segments that could possibly execute in parallel, we require that the memory mapping is set up so that one segment cannot evict a cache block that another segment has fetched to the cache. (44),(45), and (46) express that.

Let $\mathrm{mb}_{i,j,g,b}$ denote an upper bound on the number of memory accesses from the g^{th} segment of the j^{th} stage of task τ_i to memory bank b. This gives us (38). Let $\mathrm{mmb}_{i,i',j',g',b}$ be an upper bound on the number of memory accesses on memory bank b from multiple jobs of the g'^{th} segment of the j'^{th} stage of task $\tau_{i'}$ such that these memory access can impact a job of τ_i. (39) expresses it. In the proof of Theorem 1, we will show that it is an upper bound. Let $\mathrm{mmbo}_{i,i',j',g',b}$ be an

$$\mathrm{cu}_i = \sum_{j=1}^{\mathrm{nstages}_i} (\mathrm{nseg_{i,j}} \times \mathrm{cu}_{i,j}) \qquad \mathrm{etau}_i = \sum_{j=1}^{\mathrm{nstages}_i} \left(\lceil \frac{\mathrm{nseg_{i,j}}}{m} \rceil \times \mathrm{cu}_{i,j} \right) \qquad \mathrm{bspu}_{i,j} = \frac{\mathrm{cu}_{i,j}}{s} \times \lfloor \frac{\mathrm{nseg}_{i,j}}{m} \rfloor \qquad \mathrm{spu}_{i,j} = \frac{\mathrm{cu}_{i,j}}{s} \times \lceil \frac{\mathrm{nseg}_{i,j}}{m} \rceil \tag{14}$$

$$\forall \langle i', q', j', f', k \rangle \text{ such that } (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) :$$
$$(\mathrm{wi}_k = 1) \Rightarrow (t_{i',q',j',f',k} = (q'-1) \times T_{i'} + D_{i'} - ((\sum_{j'' \in [1,j']} \mathrm{spu}_{i',j''}) + f' \times \mathrm{bspu}_{i',j'+1}) \times (K/k)) \tag{15}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) :$$
$$t_{i',q',j',f',k} = (\sum_{q \in [1, P/T_i]} I_{i,q,i',q',j',f',k} \times q \times T_i) + r_{i,i',q',j',f',k} \tag{16}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) :$$
$$\sum_{q \in [0, P/T_i]} I_{i,q,i',q',j',f',k} = 1 \tag{17}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) : r_{i,i',q',j',f',k} \leq T_i \tag{18}$$

$$\forall \langle i, q, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (q \in [0, P/T_i]) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) :$$
$$(I_{i,q,i',q',j',f',k} = 1) \Rightarrow (\mathrm{aj}_{i,i',q',j',f',k} = (q+1) \times \mathrm{cu}_i) \tag{19}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) :$$
$$\mathrm{wjf}_{i,i',q',j',f',k} + (\sum_{j \in [0, \mathrm{nstages}_i - 1]} \mathrm{wjsf}_{i,j,i',q',j',f',k}) + (\sum_{j \in [0, \mathrm{nstages}_i - 1]} \mathrm{wjss}_{i,j,i',q',j',f',k}) + \mathrm{wjt}_{i,i',q',j',f',k} = 1 \tag{20}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjf}_{i,i',q',j',f',k} = 1) \Rightarrow ((D_i - r_{i,i',q',j',f',k}) \times (k/K) \leq 0) \tag{21}$$

$$\forall \langle i, j, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [0, \mathrm{nstages}_i - 1]) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjsf}_{i,j,i',q',j',f',k} = 1) \Rightarrow (\sum_{j'' \in [1,j]} \mathrm{spu}_{i,j''} \leq (D_i - r_{i,i',q',j',f',k}) \times (k/K) \leq (\sum_{j'' \in [1,j]} \mathrm{spu}_{i,j''}) + \mathrm{bspu}_{i,j+1}) \tag{22}$$

$$\forall \langle i, j, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [0, \mathrm{nstages}_i - 1]) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjss}_{i,j,i',q',j',f',k} = 1) \Rightarrow ((\sum_{j'' \in [1,j]} \mathrm{spu}_{i,j''}) + \mathrm{bspu}_{i,j+1} \leq (D_i - r_{i,i',q',j',f',k}) \times (k/K) \leq \sum_{j'' \in [1,j+1]} \mathrm{spu}_{i,j''}) \tag{23}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjt}_{i,i',q',j',f',k} = 1) \Rightarrow (\sum_{j'' \in [1, \mathrm{nstages}_i]} \mathrm{spu}_{i,j''} \leq (D_i - r_{i,i',q',j',f',k}) \times (k/K)) \tag{24}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjf}_{i,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = 0) \tag{25}$$

$$\forall \langle i, j, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [0, \mathrm{nstages}_i - 1]) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjsf}_{i,j,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = (\sum_{j'' \in [1,j]} \mathrm{nseg}_{i,j''} \times \mathrm{cu}_{i,j''}) +$$
$$((D_i - r_{i,i',q',j',f',k}) \times (k/K) - (\sum_{j'' \in [1,j]} \mathrm{spu}_{i,j''})) \times m \times s) \tag{26}$$

$$\forall \langle i, j, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [0, \mathrm{nstages}_i - 1]) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjss}_{i,j,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = (\sum_{j'' \in [1,j]} \mathrm{nseg}_{i,j''} \times \mathrm{cu}_{i,j''}) +$$
$$\mathrm{bspu}_{i,j+1} \times m \times s + ((D_i - r_{i,i',q',j',f',k}) \times (k/K) - (\sum_{j'' \in [1,j]} \mathrm{spu}_{i,j''}) - \mathrm{bspu}_{i,j+1}) \times (\mathrm{nseg}_{i,j} \bmod m) \times s) \tag{27}$$

$$\forall \langle i, i', q', j', f', k \rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge$$
$$(f' \in \{0,1\}) \wedge (k \in [1, K]) : (\mathrm{wjt}_{i,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = \mathrm{cu}_i) \tag{28}$$

$$\forall \langle i', q', j', f', k \rangle \text{ such that } (\tau_{i'} \in \tau) \wedge (q' \in [1, P/T_{i'}]) \wedge (j' \in [0, \mathrm{nstages}_{i'} - 1]) \wedge (f' \in \{0,1\}) \wedge (k \in [1, K]) :$$
$$(\mathrm{wi}_k = 1) \Rightarrow ((\sum_{\tau_i \in \tau} (\mathrm{aj}_{i,i',q',j',f',k} - w_{i,i',q',j',f',k})) \leq (m - (m-1) \times (k/K)) \times t_{i',q',j',f',k} \times s) \tag{29}$$

$$\forall k \text{ such that } k \in [1, K] : (\mathrm{wi}_k = 1) \Rightarrow ((\sum_{\tau_i \in \tau} \mathrm{cu}_i / T_i) \leq (m - (m-1) \times (k/K)) \times s) \tag{30}$$

$$\forall \langle i, k \rangle \text{ such that } (\tau_i \in \tau) \wedge (k \in [1, K]) : (\mathrm{wi}_k = 1) \Rightarrow (\mathrm{etau}_i \leq (k/K) \times s \times D_i) \tag{31}$$

$$\sum_{k=1}^{K} \mathrm{wi}_k \geq 1 \tag{32}$$

$$\mathrm{cu}_{i,j} \in \mathbb{R}_{\geq 0}, \mathrm{cu}_i \in \mathbb{R}_{\geq 0}, \mathrm{etau}_i \in \mathbb{R}_{\geq 0}, \mathrm{bspu}_{i,j} \in \mathbb{R}_{\geq 0}, \mathrm{spu}_{i,j} \in \mathbb{R}_{\geq 0}, t_{i',q',j',f',k} \in \mathbb{R}_{\geq 0}, I_{i,q,i',q',j',f',k} \in \{0,1\}, r_{i,i',q',j',f',k} \in \mathbb{R}_{\geq 0},$$
$$\mathrm{aj}_{i,i',q',j',f',k} \in \mathbb{R}_{\geq 0}, \mathrm{wjf}_{i,i',q',j',f',k} \in \{0,1\}, \mathrm{wjsf}_{i,j,i',q',j',f',k} \in \{0,1\}, \mathrm{wjss}_{i,j,i',q',j',f',k} \in \{0,1\}, \mathrm{wjt}_{i,i',q',j',f',k} \in \{0,1\},$$
$$\mathrm{w}_{i,i',q',j',f',k} \in \mathbb{R}_{\geq 0}, \mathrm{wi}_k \in \{0,1\} \tag{33}$$

Fig. 3: Schedulability analysis for gEDF scheduling of parallel tasks formulated as a MILP.

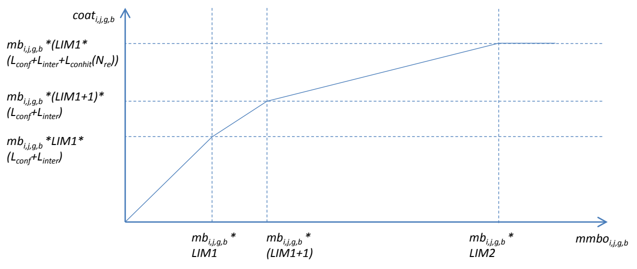


Fig. 4: Contention on bank queue. The vertical axis shows an upper bound on the delay that the $\mathrm{mb}_{i,j,g,b}$ memory accesses from the g^{th} segment of the j^{th} stage of task τ_i can suffer from because the other $\mathrm{mmbo}_{i,j,g,b}$ memory accesses from other segments access memory of bank b and hence contend on the queue for memory bank b.

upper bound on the number of memory accesses on memory bank b from multiple jobs of all other segments than the g^{th} segment of the j^{th} stage of task τ_i such that these memory accesses can impact a job of the g^{th} segment of the j^{th} stage of task τ_i. (40) expresses it.

Now consider memory contention. Look at the queues inside the memory controller in Fig. 1. Consider the g^{th} segment of the j^{th} stage of task τ_i and its (at most $\mathrm{mb}_{i,j,g,b}$) memory accesses that it performs on memory locations of memory bank b. Let $\mathrm{coat}_{i,j,g,b}$ denote an upper bound on the extra execution time of this segment because other memory accesses (from other segments) access this bank (bank b). Let $\mathrm{oao}_{i,j,g,b}$ denote an upper bound on the number of other memory accesses that causes extra execution time of this segment because other memory accesses (from other segments) access other banks (than bank b). Then, we express $\mathrm{cm}_{i,j,g}$ as

$$\mathrm{cm}_{i,j,g} = \mathrm{C}_{i,j} + s \times (\sum_{b=0}^{B-1} (\mathrm{coat}_{i,j,g,b} + L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b})) \tag{35}$$

In the above equation, we multiply by s because $\mathrm{coat}_{i,j,g,b}$ and $L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b}$ measure execution time whereas $\mathrm{cm}_{i,j,g}$ measures execution requirement.

We will now find expressions for $\mathrm{coat}_{i,j,g,b}$ and $\mathrm{oao}_{i,j,g,b}$. For this purpose, look again at the queues inside the memory controller in Fig. 1. A single memory access accessing bank b can be delayed by the following:

1) There are already memory accesses in the queue of bank b when this single memory access is inserted in the queue of bank b and because of FIFO queuing, these other memory accesses are served first.
2) After this single memory access is enqueued in the queue for bank b, there are other memory accesses enqueued in this bank and these other memory accesses' row is currently loaded in the row buffer and hence they get ahead in the queue for this memory bank (reordering).
3) When one of the other memory accesses mentioned in 1) or 2) reaches the head of the queue of bank b, it is not served immediately; instead it has to wait for the memory bus being granted and this takes time because other memory accesses in the queues to other memory banks than bank b use the memory bus.

About 1) and 2) Since we assume a processor stalls until its memory access has been completed, it follows that from each processor, there can be at most one outstanding memory access and hence there are at most LIM1 memory accesses of 1) above. The hardware places a limit on the number of reorderings that can happen. In previous work [13], we introduced the parameter to indicate an upper bound on the number of those reorderings that a single memory access can experience. In this paper, we let N_{re} denote this parameter; a typical value [13] is $N_{\mathrm{re}} = 12$. Consequently, the $\mathrm{mb}_{i,j,g,b}$ memory accesses from the g^{th} segment of the j^{th} stage of task τ_i performing on bank b has to wait for at most $\mathrm{mb}_{i,j,g,b} \times (\mathrm{LIM1} + N_{\mathrm{re}}) = \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM2}$ other memory accesses performing on bank b (because of 1) and 2) above). Because $\mathrm{mmb}_{i,i',j',g',b}$ is an upper bound on the number of memory accesses from the g'^{th} segment of the j'^{th} stage of task $\tau_{i'}$ that can be performed in parallel with a segment of a job of task τ_i, assuming that the job of task τ_i meets its deadline, it follows, using (40), that: The $\mathrm{mb}_{i,j,g,b}$ memory accesses from the g^{th} segment of the j^{th} stage of task τ_i performing on bank b has to wait for at most

$$\min(\mathrm{mb}_{i,j,g,b} \times \mathrm{LIM2}, \mathrm{mmbo}_{i,j,g,b}) \tag{36}$$

other memory accesses performing on bank b (because of 1) and 2) above). Let $\mathrm{oat}_{i,j,g,b}$ be the expression in (36). (It means other accesses to this bank.) By inspecting $L_{\mathrm{conhit}}(x)$ and the parameters in Section II, one can see that these memory accesses have different effects; the memory accesses that are in the queue before a memory access has arrived to the queue cause more interference than the ones that arrive later that cause reordering. Fig. 4 shows an upper bound.

About 3) A memory access related to memory bank b is inserted in the queue for the memory bus only if (i) this memory access is at the head of the queue of the memory bank b and (ii) there is no memory access related to memory bank b already in the queue of the memory bus. Hence, a memory access that has reached the head of the queue of its memory bank needs to wait for at most B-1 other memory accesses until it is granted the memory bus. Consequently, the $\mathrm{mb}_{i,j,g,b}$ memory accesses from the g^{th} segment of the j^{th} stage of task τ_i performing on bank b has to wait for at most $(\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1)$ other memory accesses performing on bank b (because of 3)). Because $\mathrm{mmb}_{i,i',j',g',b}$ is an upper bound on the number of memory accesses from the g'^{th} segment of the j'^{th} stage of task $\tau_{i'}$ that can be performed in parallel with a segment of a job of task τ_i, assuming that the job of task τ_i meets its deadline, it follows, using (40), that: The $\mathrm{mb}_{i,j,g,b}$ memory accesses from the g^{th} segment of the j^{th} stage of task τ_i performing on bank b has to wait for at most

$$\min((\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1), \sum_{b'' \in [0,B-1] \wedge (b'' \neq b)} \mathrm{mmbo}_{i,j,g,b''}) \tag{37}$$

other memory accesses performing on other banks than bank b (because of 3) above).

This reasoning yields an upper bound on the execution requirement of a segment on a form close to MILP — see Fig. 5.

V. The MILP formulation

Let Π denote the computer platform (the parameters m, s, H, B and the parameters describing the memory system). $\mathrm{fmem}(\tau, \Pi, K)$ is a function which returns the tuple $\langle \mathrm{flag}, o \rangle$ where flag is a boolean and o is a multi-dimensional array. If there exists an assignment of values to the variables so that the constraints in Fig. 3 and Fig. 5 are satisfied then flag is

$$\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : \mathrm{mb}_{i,j,g,b} = \sum_{p=0}^{\mathrm{np}_{i,j}-1} \sum_{h=0}^{H-1} \mathrm{MA}_{i,j,p} \times \mathrm{o}_{i,j,g,p,h,b} \tag{38}$$

$$\begin{aligned}&\forall\langle i,i',j',g',b\rangle \text{ such that } (\tau_i \in \tau) \wedge (\tau_{i'} \in \tau) \wedge (j' \in [1,\mathrm{nstages}_{i'}]) \wedge (g' \in [1,\mathrm{nseg}_{i',j'}]) \wedge (b \in [0,B-1]) : \\ &\mathrm{mmb}_{i,i',j',g',b} = (\lceil \frac{D_i}{T_i'} \rceil + 1) \times \mathrm{mb}_{i',j',g',b}\end{aligned} \tag{39}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : \\ &\mathrm{mmbo}_{i,j,g,b} = \sum_{\tau_{i'} \in \tau} \sum_{j' \in [1,\mathrm{nstages}_{i'}]} \sum_{g' \in [1,\mathrm{nseg}_{i',j'}] \wedge (((i'=i) \wedge (j'=j) \wedge (g' \neq g)) \vee (i' \neq i))} \mathrm{mmb}_{i,i',j',g',b}\end{aligned} \tag{40}$$

$$\forall\langle i,j,g,p\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (p \in [0,\mathrm{np}_{i,j}-1]) : \sum_{h=0}^{H-1} \sum_{b=0}^{B-1} \mathrm{o}_{i,j,g,p,h,b} = 1 \tag{41}$$

$$\forall\langle h,b\rangle \text{ such that } (h \in [0,H-1]) \wedge (b \in [0,B-1]) : (\sum_{\tau_i \in \tau} \sum_{j \in [1,\mathrm{nstages}_i]} \sum_{g \in [1,\mathrm{nseg}_{i,j}]} \sum_{p \in [0,\mathrm{np}_{i,j}-1]} (\frac{1}{\mathrm{GS}_{i,j,g,p}} \times \mathrm{o}_{i,j,g,p,h,b})) + \mathrm{ino}_{h,b} \leq \mathrm{CAP} \tag{42}$$

$$\sum_{h \in [0,H-1]} \sum_{b \in [0,B-1]} \mathrm{ino}_{h,b} = \mathrm{INO} \tag{43}$$

$$\forall\langle i,j,g,h\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (h \in [0,H-1]) : (\mathrm{x}_{i,j,g,h} = 1) \Rightarrow (\sum_{p \in [0,\mathrm{np}_{i,j}-1]} \sum_{b \in [0,B-1]} \mathrm{o}_{i,j,g,p,h,b} \geq 1) \tag{44}$$

$$\forall\langle i,j,g,h\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (h \in [0,H-1]) : (\mathrm{x}_{i,j,g,h} = 0) \Rightarrow (\sum_{p \in [0,\mathrm{np}_{i,j}-1]} \sum_{b \in [0,B-1]} \mathrm{o}_{i,j,g,p,h,b} \leq 0) \tag{45}$$

$$\begin{aligned}&\forall\langle i,j,g,i',j',g',h\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (\tau_{i'} \in \tau) \wedge (j' \in [1,\mathrm{nstages}_{i'}]) \wedge (g' \in [1,\mathrm{nseg}_{i',j'}]) \wedge \\ &(h \in [0,H-1]) \wedge (((i=i') \wedge (j=j') \wedge (g<g')) \vee (i<i')) : \mathrm{x}_{i,j,g,h} + \mathrm{x}_{i',j',g',h} \leq 1\end{aligned} \tag{46}$$

$$\forall\langle i,j,g\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) : \mathrm{cm}_{i,j,g} \leq \mathrm{cu}_{i,j} \tag{47}$$

$$\forall\langle i,j\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) : \sum_{g=1}^{\mathrm{nseg}_{i,j}} \mathrm{se}_{i,j,g} = 1 \tag{48}$$

$$\forall\langle i,j,g\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) : (\mathrm{se}_{i,j,g} = 1) \Rightarrow (\mathrm{cm}_{i,j,g} \geq \mathrm{cu}_{i,j}) \tag{49}$$

$$\begin{aligned}&\forall\langle i',j',g',p',i'',j'',g'',p'',h,b\rangle \text{ such that } (\langle i',j',g',p',i'',j'',g'',p''\rangle \in \mathrm{sharedframes}) \wedge (h \in [0,H-1]) \wedge (b \in [0,B-1]) : \\ &o_{i',j',g',p',h,b} = o_{i'',j'',g'',p'',h,b}\end{aligned} \tag{50}$$

$$\forall\langle i,j,g\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) : \mathrm{cm}_{i,j,g} = \mathrm{C}_{i,j} + s \times (\sum_{b=0}^{B-1} (\mathrm{coat}_{i,j,g,b} + L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b})) \tag{51}$$

$$\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : \mathrm{bc1}_{i,j,g,b} + \mathrm{bc2}_{i,j,g,b} + \mathrm{bc3}_{i,j,g,b} + \mathrm{bc4}_{i,j,g,b} = 1 \tag{52}$$

$$\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc1}_{i,j,g,b} = 1) \Rightarrow (\mathrm{mmbo}_{i,j,g,b} \leq \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM1}) \tag{53}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc2}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{mb}_{i,j,g,b} \times \mathrm{LIM1} \leq \mathrm{mmbo}_{i,j,g,b} \leq \mathrm{mb}_{i,j,g,b} \times (\mathrm{LIM1}+1))\end{aligned} \tag{54}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc3}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{mb}_{i,j,g,b} \times (\mathrm{LIM1}+1) \leq \mathrm{mmbo}_{i,j,g,b} \leq \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM2})\end{aligned} \tag{55}$$

$$\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow (\mathrm{mb}_{i,j,g,b} \times \mathrm{LIM2} \leq \mathrm{mmbo}_{i,j,g,b}) \tag{56}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc1}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{coat}_{i,j,g,b} = \mathrm{mmbo}_{i,j,g,b} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}))\end{aligned} \tag{57}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc2}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM1} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + (\mathrm{mmbo}_{i,j,g,b} - \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM1}) \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}}) \times t_{CK})\end{aligned} \tag{58}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc3}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM1} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + \mathrm{mb}_{i,j,g,b} \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}}) \times t_{CK} + \\ &(\mathrm{mmbo}_{i,j,g,b} - \mathrm{mb}_{i,j,g,b} \times (\mathrm{LIM1}+1)) \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}} + \mathrm{CL}) \times (1/2) \times t_{CK})\end{aligned} \tag{59}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times (\mathrm{LIM1} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + L_{\mathrm{conhit}}(N_{\mathrm{re}})))\end{aligned} \tag{60}$$

$$\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc4}_{i,j,g,b} = 0) \Rightarrow (\mathrm{oat}_{i,j,g,b} = \mathrm{mmbo}_{i,j,g,b}) \tag{61}$$

$$\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow (\mathrm{oat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LIM2}) \tag{62}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bu}_{i,j,g,b} = 0) \Rightarrow \\ &(\sum_{b' \in [0,B-1] \wedge (b' \neq b)} \mathrm{mmbo}_{i,j,g,b'} \leq (\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1))\end{aligned} \tag{63}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bu}_{i,j,g,b} = 1) \Rightarrow \\ &(\sum_{b' \in [0,B-1] \wedge (b' \neq b)} \mathrm{mmbo}_{i,j,g,b'} \geq (\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1))\end{aligned} \tag{64}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bu}_{i,j,g,b} = 0) \Rightarrow \\ &(\mathrm{oao}_{i,j,g,b} = (\sum_{b' \in [0,B-1] \wedge (b' \neq b)} \mathrm{mmbo}_{i,j,g,b'}))\end{aligned} \tag{65}$$

$$\begin{aligned}&\forall\langle i,j,g,b\rangle \text{ such that } (\tau_i \in \tau) \wedge (j \in [1,\mathrm{nstages}_i]) \wedge (g \in [1,\mathrm{nseg}_{i,j}]) \wedge (b \in [0,B-1]) : (\mathrm{bu}_{i,j,g,b} = 1) \Rightarrow \\ &(\mathrm{oao}_{i,j,g,b} = (\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1))\end{aligned} \tag{66}$$

$$\begin{aligned}&\mathrm{mb}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}, \mathrm{mmb}_{i,i',j',g',b} \in \mathbb{Z}_{\geq 0}, \mathrm{mmbo}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}, \mathrm{o}_{i,j,g,p,h,b} \in \{0,1\}, \mathrm{ino}_{h,b} \in \mathbb{Z}_{\geq 0}, \mathrm{x}_{i,j,g,h} \in \{0,1\}, \mathrm{cm}_{i,j,g} \in \mathbb{R}_{\geq 0}, \mathrm{se}_{i,j,g} \in \{0,1\}, \\ &\mathrm{b1}_{i,j,g,b} \in \{0,1\}, \mathrm{b2}_{i,j,g,b} \in \{0,1\}, \mathrm{b3}_{i,j,g,b} \in \{0,1\}, \mathrm{b4}_{i,j,g,b} \in \{0,1\}, \mathrm{bu}_{i,j,g,b} \in \{0,1\}, \mathrm{coat}_{i,j,g,b} \in \mathbb{R}_{\geq 0}, \mathrm{oao}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}, \mathrm{oat}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}\end{aligned} \tag{67}$$

Fig. 5: Expressing increased execution time.

true and o is the values of the o-variables; otherwise flag is false and o is undefined.

Theorem 1.

$$((\langle \text{flag}, o\rangle = \text{fmem}(\tau, \Pi, K)) \land (\text{flag} = \text{true})) \Rightarrow$$
$$\tau \text{ is gEDF schedulable on a computer with } m \text{ processors of speed } s \text{ for the case that tasks experience memory contention and the memory mapping is compatible with } o$$

Proof: If the theorem is false then there exists a τ, m, s, K and an assignment of the number of jobs that each task generates and an assignment of arrival time to jobs and execution requirement of segments and a schedule such that the following two statements are true:

1) $(\langle \text{flag}, o\rangle = \text{fmem}(\tau, \Pi, K)) \land (\text{flag} = \text{true})$
2) for the jobset generated by τ with the aforementioned assignment, it holds that gEDF can generate the aforementioned schedule and there is at least one job that misses its deadline in this schedule.

For this schedule, let t_0 denote the earliest time when a deadline miss occurs. Remove all jobs with arrival time $\geq t_0$. There is still a deadline miss at time t_0. Let us now reason as follow: For each job with absolute deadline $> t_0$ such that it performs execution after time t_0, do the following: identify the latest stage of this job such that there is a segment of this stage that performs execution after t_0. Then reduce the execution of this segment. Repeated application of this yields that no job with absolute deadline $> t_0$ performs execution after time t_0. Hence, it holds that: (i) 1) and 2) above are true, (ii) one or many jobs with absolute deadline at t_0 misses deadlines, (iii) each job with absolute deadline $< t_0$ meets its deadline, (iv) all jobs have arrival times $< t_0$, and (v) no job with absolute deadline $> t_0$ performs execution after time t_0.

For each job with absolute deadline $< t_0$, we can reason as follows: Let τ_i denote the task that generates the job. Let A denote the arrival time of this job and consider the time interval $[A, A + D_i)$ and consider a task i' which is not the task that generated the job of task τ_i. Because of (iii) and (iv), there can be at most one job of task i' such that this job arrives before A and it has execution that overlaps with $[A, A + D_i)$. Also, because of (iii), there can be at most $\lceil D_i/T_{i'} \rceil$ jobs of task i' such that this job arrives at or after A and it has execution that that overlaps with $[A, A + D_i)$.

For each job with absolute deadline $\geq t_0$, we can reason as follows: Let τ_i denote the task that generates the job. Let A denote the arrival time of this job and consider the time interval $[A, A + D_i)$ and consider a task i' which is not the task that generated the job of task τ_i. Because of (iii) and (iv), there can be at most one job of task i' such that this job arrives before A and it has execution that overlaps with $[A, A + D_i)$. Also, because of (v), there can be at most $\lceil D_i/T_{i'} \rceil$ jobs of task i' such that this job arrives at or after A and it has execution that that overlaps with $[A, A + D_i)$.

Consequently, for each of these cases, there are at most $(\lceil \frac{D_i}{T'_i} \rceil + 1) \times \text{mb}_{i',j',g',b}$ memory accesses on bank b of jobs of the g'^{th} segment of the j'^{th} stage of task τ'_i that overlaps with $[A, A + D_i)$. This expression is the right-hand side of the expression of (39). Hence, there are at most $\text{mmb}_{i,i',j',g',b}$ memory accesses of jobs of the g'^{th} segment of the j'^{th} stage of task τ'_i that overlaps with $[A, A + D_i)$. Since we know the values of $\text{mmb}_{i,i',j',g',b}$, using Fig. 5 yields $\text{cm}_{i,j,g}$. This yields $\text{cu}_{i,j}$ which provides an upper bound on the execution requirement. Since $\text{cu}_{i,j}$ is an upper bound on execution requirement we can treat the system as if there was no contention for resources in the memory system and execution requirements were given by $\text{cu}_{i,j}$. Since the constraints in Fig. 3 are satisfied, all deadlines are met. This contradicts 2) above. Hence, the theorem is correct. ■

Note that some of the constraints mentioned are not MILP — they have binary variables and logical operators. We will discuss this now. A constraint of the form $(x = 1) \Rightarrow (a = b)$ can be rewritten as: $((x = 1) \Rightarrow (a \leq b)) \land ((x = 1) \Rightarrow (a \geq b))$. Note that if x is a variable with the domain $\{0, 1\}$ and a and b are non-negative real variables and BIG is a constant selected so that $a \leq \text{BIG}$ and $b \leq \text{BIG}$, then a constraint $(x = 1) \Rightarrow (a \leq b)$ can be rewritten as

$$a - b + \text{BIG} \times x \leq \text{BIG} \tag{68}$$

Note that in a feasible solution to Fig. 3 and Fig. 5, for the variables in the constraints (52)-(67), the variable is at most

$$\max_{\tau_i \in \tau}\Big(\sum_{\tau'_i \in \tau} ((\lceil \frac{D_i}{T'_i} \rceil + 1) \times (\sum_{j \in [1', \text{nstages}_{i'}]} \sum_{g' \in [1, \text{nseg}_{i',j'}]} \sum_{p' \in [0, \text{np}_{i',j'} - 1]} \text{MA}_{i',j',p'}))\Big) \tag{69}$$

Hence, for the constraints (52)-(67), the left-hand side (lhs) is at most

$$\max\Big(2 \times (B - 1), L_{\text{conf}} + L_{\text{inter}}\Big) \times ((69)) \tag{70}$$

Also, for each of the other constraints, the lhs is at most

$$(P + \text{DMAX}) \times m \times \max(1, s) + H \times B \tag{71}$$

Applying the rewriting expressed by (68) (and minor variants of it), with $\text{BIG} = \max((70), (71))$, yields that all of our constraints can be converted to a MILP.

VI. Discussion

Recall (from Section II) that in general, it is possible to obtain (e.g. using a worst-case execution-time analysis tool) $C_{i,j}(\text{map})$ and $\text{MA}_{i,j,p}(\text{map})$ but it is very expensive to obtain $C_{i,j}$ and $\text{MA}_{i,j,p}$. This can be dealt with by guessing values of the latter and call the function $\text{fmem}(\tau, \Pi, K)$ and then obtain a new memory mapping and then for this memory mapping, check whether the guess was valid. Also, note that solving the MILP produces an abstract memory mapping o. It is abstract because it does not specify exactly to which memory frame a page should be mapped; it only specifies to which cache color and bank color a memory page should be mapped. We assume a method exists that converts the abstract mapping o to a mapping map that specifies for each page which frame it maps to (it is trivial to create it). An algorithm based on these ideas is shown below:

```
1)  Choose a value of $K$ (for example $K = 20$)
2)  Choose a value of maxiter (for example maxiter = 3)
3)  Choose one abstract memory mapping $o'$
4)  **for** iter := 1 to maxiter **do**
5)    choose a memory mapping $\text{map}'$ that is compatible with the
6)      abstract memory mapping $o'$
7)    $\forall i, j$ :
8)      obtain $C_{i,j}(\text{map}')$ and then assign $C_{i,j}^{\text{guess}} := C_{i,j}(\text{map}')$
9)    $\forall i, j, p$ :
10)     obtain $\text{MA}_{i,j,p}(\text{map}')$ then assign $\text{MA}_{i,j,p}^{\text{guess}} := \text{MA}_{i,j,p}(\text{map}')$
11)   $\langle \text{flag}, o\rangle = \text{fmem}(\tau, \Pi, K)$; in this call, assume that
12)     $\forall i, j : C_{i,j} = C_{i,j}^{\text{guess}}$; $\forall i, j, p : \text{MA}_{i,j,p} = \text{MA}_{i,j,p}^{\text{guess}}$
13)   **if** flag **then**
14)     choose a memory mapping map that is compatible with the
15)       abstract memory mapping $o$
```

```
16)         o' := o
17)         **if** $(\forall i,j : C_{i,j}(\text{map}) \le C^{\text{guess}}_{i,j})$ **and**
18)           $(\forall i,j,p : \text{MA}_{i,j,p}(\text{map}) \le \text{MA}^{\text{guess}}_{i,j,p})$ **then**
19)             declare SUCCESS
20)         **end if**
21)     **else**
22)         Choose one abstract memory mapping $o'$ that has not been
23)           tried before
24)     **end if**
25) **end for**
26) declare FAILURE
```

Hence, our solution can be used in practice.

We have implemented a tool based on this theory and tested it on systems with 4 and 8 processors. With this experimentation, we find that such systems can be analyzed and configured and the pessimism is reasonable. See appendix.

VII. Conclusions

COTS multicore processors are the norm today but their use for hard real-time systems is challenging because (i) in order to take full advantage of such platforms for meeting tight deadlines, parallelization is necessary and (ii) the contention for shared resources in the memory system makes execution times hard to predict. In this paper, we have developed a solution that addresses these issues. Our main idea is to formulate a MILP that configures the memory mapping and performs schedulability analysis.

Acknowledgment

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0001727

References

[1] B. Andersson and D. de Niz. Analyzing Global-EDF for multiprocessor scheduling of parallel tasks. In *Proc. of OPODIS*, 2012.
[2] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Döbel, and H. Härtig. Response-time analysis of parallel fork-join workloads with real-time constraints. In *Proc. of ECRTS*, 2013.
[3] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *Proc. of RTSS*, 2012.
[4] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG model. In *Proc. of ECRTS*, 2013.
[5] H. S. Chwa, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *Proc. of ECRTS*, 2013.
[6] S. Collette, L. Cucu, and J. Goossens. Integrating job parallelism in real-time scheduling theory. *Information Processing Letters*, 2008.
[7] L. Cong and J. H. Anderson. Supporting soft real-time dag-based systems on multiprocessors with no utilization loss. In *Proc. of RTSS*, 2010.
[8] D. Dasari, B. Andersson, V. Nélis, S. M. Petters, A. Easwaran, and J. Lee. Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus. In *ICESS'11*.
[9] F. Fauberteau, S. Midonnet, and M. Qamhieh. Partitioned scheduling of parallel real-time tasks on multiprocessor systems. *SIGBED Review*, 2011.
[10] D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu. A real-time scheduling service for parallel tasks. In *Proc. of RTAS*, 2013.
[11] C. Han and K.-J. Lin. Scheduling Parallelizable jobs on multiprocessors. In *Proc. of RTSS*, 1989.
[12] S. Kato and Y. Ishikawa. Gang EDF scheduling of parallel task systems. In *Proc. of RTSS*, 2009.
[13] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding Memory Interference Delay in COTS-based Multi-Core Systems. In *Proc. of RTAS*, 2014.
[14] H. Kim, A. Kandhalu, and R. Rajkumar. A Coordinated Approach for Practical OS-Level Cache Management in Multi-Core Real-Time Systems. In *Proc. of ECRTS*, 2013.
[15] J. Kim, H. Kim, K. Lakshmanan, and R. R. Rajkumar. Parallel scheduling for cyber physical systems: Analysis and case study on a self-driving car. In *Proc. of RTAS*, 2013.
[16] K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *Proc. of RTSS*, 2010.
[17] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of global EDF for parallel tasks. In *Proc. of ECRTS*, 2013.
[18] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. PACT'12.
[19] M. Lv, G. Nan, W. Yi, and G. Yu. Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In *Proc. of RTSS*, 2010.
[20] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-Time Cache Management Framework for Multi-core Architectures. In *Proc. of RTAS*, 2013.
[21] L. Nogueira and L. P. Pinho. Server-based scheduling of parallel real-time tasks. In *Proc. of EMSOFT*, 2012.
[22] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *DATE'10*.
[23] M. Qamhieh, F. Fauberteau, G. Laurent, and S. Midonnet. Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In *Proc. of RTNS*, 2013.
[24] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation. In *CODES+ISSS'11*.
[25] J. Rosén, A. Andrei, P. Eles, and Z. Peng. Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *Proc. of RTSS*, 2007.
[26] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel tasks models. In *Proc. of RTSS*, 2011.
[27] L. Steffens, M. Agarwal, and P. van der Wolf. Real-Time Analysis for Memory Access in Media Processing SoCs - A Practical Approach. In *Proc. of ECRTS*, 2008.
[28] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar. Coordinated bank and cache coloring for temporal protection of memory accesses. In *ICESS'13*.
[29] Q. Wang and G. Parmer. FJOS: Practical, predictable, and efficient system support for fork-join parallelism. In *Proc. of RTAS*, 2014.
[30] B. Ward, J. Herman, C. Kenna, and J. Anderson. Making Shared Caches More Predictable on Multicore Platforms. In *Proc. of ECRTS*, 2013.
[31] Z. P. Wu, Y. Krish, and R. Pellizzoni. Worst Case Analysis of DRAM Latency in Multi-Requestor Systems. In *Proc. of RTSS*, 2013.
[32] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory Access Control in Multiprocessor for Real-time Systems with Mixed Criticality. In *Proc. of ECRTS*, 2012.